The SpeedChex Web Service API for Merchants
Remote Deposit Commands
Version 1.0

Introduction

The Remote Deposit Commands detailed in this API document give Merchants a comprehensive and feature-rich solution that facilitates the scanning and "remote deposit" of paper checks into merchant bank accounts. Scanned checks submitted through this API to the SpeedChex Gateway will be converted into either ACH or Check 21 (IRD) transactions and processed to the Federal Reserve for deposit.

To submit checks for remote deposit through this API, the paper checks must be scanned using a scanner that images both the front and back sides of the check at a minimum resolution of 200 dpi. The scanner must also capture MICR information (the routing number, account number, and check number) from the bottom of each check

Merchants have the option to use the SpeedChex Gateway as their storage medium for all data and images captured from each check. The transaction-level management commands defined in this API allow your software to interact with the stored check data allowing your users to retrieve, review and update each remote deposit transaction until all transactions in a batch are ready for deposit.

Traditionally, users processing scanned checks would review and manually input non-scanned check information like the amount (required) or the payer name (optional) or manually correct scanning errors. The SpeedChex Gateway offers a technology called ***SpeedChex SmartScan*** which performs image character recognition to capture and/or correct information from the check image. A detailed list of the features available from ***SpeedChex SmartScan*** product are available later in this document.

The SpeedChex Gateway and the Command/Response System

The SpeedChex Gateway supports procesing for multiple payment methods including credit cards, ATM pin-debit cards, electronic checks (ACH), and remote deposit of scanned checks (Check 21).

Software applications can communicate with the SpeedChex Gateway through multiple, established Internet protocols, including the following:

- SOAP 1.2 Web Services w/MTOM attachment support
- Microsoft WCF Web Services (in development)
- Traditional HTTP POST

Although each of these communication protocols requires API documentation that is specific to the functionality of that protocol, each API shares a common command/response method for interacting with the gateway.

Simply put, your software issues a command to the SpeedChex Gateway to accomplish any specific task and the gateway will send back a formatted response to your command.

The Command/Response system is quite extensive and supports the ability to perform a variety of transaction management tasks including:

- Creating/Authorizing new payment transactions
- Uploading batches of transactions
- Modifying/Cancelling existing transactions or batches
- Retrieving reports
- Querying payment data

This document, like all other API documents for the SpeedChex Gateway, is targeted toward a specific subset of commands that are grouped according to either payment method, the task(s) to be performed, or both.

Remote Deposit (Check 21) Commands

The following is a list and brief explanation of the Remote Deposit Commands that can be issued through the ***SpeedChex Web Service API:***

- **RemoteDepositBatch.CreateNewBatch** – Creates a new Remote Deposit Batch for placing, managing and utlimately processing scanned checks for remote deposit.
- **RemoteDepositBatch.AddTransaction** – Adds a check to a new or existing Remote Deposit Batch.
- **RemoteDepositBatch.UploadBatch** – Executes an array of *RemoteDepositBatch.AddTransaction* commands allowing you to add multiple transactions in "batch mode" to a new or existing Remote Deposit Batch. This command would be equivalent to sending a R*emoteDepositBatch.CreateNewBatch* command and then sending multiple R*emoteDepositBatch.AddTransaction* commands each separately
- **RemoteDepositBatch.MarkForDeposit** – Changes the state of a Remote Deposit Batch to 'Mark for Deposit' causing it to process on the next process cut-off for the specified scheduled deposit date.
- **RemoteDepositBatch.HoldForReview** – Changes the state of a Remote Deposit Batch to 'Hold for Review' which holds the batch for user review/edit until it is ready for deposit
- **RemoteDepositBatch.VoidBatch** – Cancels a Remote Deposit Batch that has not yet been sent to the Federal Reserve. All transaction data and images associated with the batch will be deleted permanently.
- ***RemoteDepositBatch.ModifyTransaction*** *– Modifies any element of a transaction as long as the batch containing the transaction is still in Review mode (has not yet been deposited).*
- ***RemoteDepositBatch.RemoveTransaction*** *– Removes a transaction from a batch as long as the batch containing the transaction is still in Review mode (has not yet been deposited).*
- ***RemoteDepositBatch.ReviewAllTransactions*** *– Downloads all transactions from a batch for review. Only transactions from a batch that is still in Review mode can be modified or removed.*
- ***RemoteDepositBatch.ReviewSingleTransaction*** *– Downloads a single transaction for review. Only transactions from a batch that is still in Review mode can be modified or removed.*

Please note: Greyed-out commands will be implemented in a future update of this API.

Conceptualizing Remote Deposit Batches

Paper checks can be scanned and sent individually in real-time or combined and sent with other scanned checks as a batch upload. Regardless of how scanned check transactions are sent, each remote deposit transaction must be assigned to a conceptual Remote Deposit Batch for processing.

A new Remote Deposit Batch can be created in one of three ways:

1. Sending a *RemoteDepositBatch.CreateNewBatch* command with a new, unique BatchID. This method is helpful when you want to explicity initialize a Remote Deposit Batch with a certain batch state without loading transactions yet.

2. Sending a *RemoteDepositBatch.AddTransaction* command that specifies a previously unused BatchID value. This method is helpful if you submit Remote Deposit transactions in real-time and you want the first transaction of the day to implicity create the Remote Deposit Batch.

3. Sending a *RemoteDepositBatch.UploadBatch* command that specifies a previously unused BatchID value. This method is used when you want to implicity create the Remote Deposit Batch when you upload a batch of Remote Deposit transactions.

From that point forward, you can add single transactions or batches of transactions to an existing Remote Deposit Batch by simply specifying the same Batch ID of the existing Remote Deposit Batch. Of course, if a batch has been sent to the Federal Reserve for processing, you will no longer be able to append transactions to that batch.

Remote Deposit Batch States

It is the nature of the remote deposit process that a person must review each scanned check for data errors and supply additional information that was not captured by the scanner.

This API has been designed to facilitate this process of reviewing and updating data for remote deposit transactions. As a result, a Remote Deposit Batch can be assigned one of the following states:

Possible Remote Deposit Batch States

Batch State	Description
Hold for Review	Batch will be held indefinately so users can review and update transaction information as needed.
Mark for Deposit	Batch will be scheduled for processing (deposited) at the next processing cut-off time based on the batch deposit date specified. Transactions can still be reviewed and updated until the batch is processed.
Deposited	Batch has been sent to Federal Reserve for processing and is now closed. Transactions can be reviewed but not changed.
Cancelled	Batch was manually cancelled and will not be processed unless the batch state is modified

A Remote Deposit Batch should be assigned the state of 'Hold for Review' when one or more of the following scenarios is true:

- When users still need to review, update or confirm accuracy of scanned check data.
- When taking advantage of the *SpeedChex SmartScan* technology to capture data that must be reviewed and updated or confirmed for accuracy.
- When uploading multiple transactions or batches to the same Remote Deposit Batch over time.

Once a Remote Deposit Batch in 'Hold for Review' state is assumed to be complete and accurate, a command can be issued to change the batch state to 'Mark for Deposit' which will cause the batch to be processed.

Many Merchants will have their own facilities for storing, reviewing and updating transaction data and images and may opt not to use *SpeedChex SmartScan* technology. In this instance, the 'Hold for Review' state is not needed. The Merchant would simply create a Remote Deposit Batch, add the transactions to the batch, and assign the batch state to 'Mark for Deposit'.

Please note that an error will be returned if you attempt to change the state of a Remote Deposit Batch to 'Mark for Deposit' or upload an entire batch at once with a batch state of 'Mark for Deposit' and required data is missing from any transaction in the batch. Similarly, an error will occur if the state of a Remote Deposit Batch is already set to 'Mark for Deposit' and you attempt to add more transactions to the batch.

Transactions in batches of any state can be downloaded for review at any time, but only transactions in a Remote Deposit Batch with the batch state set to 'Hold for Review' can be modified or cancelled.

Taking Advantage of SpeedChex SmartScan Technology

<content will be added in future update of this API>

Important Rules Regarding Customer Notification and SEC Codes

There are three types of check conversion that can be implemented by a business and each method has a specific three-letter code assigned to it called the SEC Code:

- **Back Office Conversion** – Allows businesses to collect checks written at a business payment counter or point-of-sale and convert them to electronic payments later in a centralized location. The SEC Code is **BOC**.

- **Point of Purchase** – Used for checks written at a point of sale, voided, returned immediately to the customer, and processed as a converted check. The SEC Code is **POP**.

- **Accounts Receivable Entry** – Mailed checks used for bill payment that are converted into electronic payments by the billing company. The SEC Code is **ARC**.

The banking industry in cooperation with the federal and state governments has setup very specific rules regarding the implementation of these check conversion methods in a business environment. These rules address (1) the notification to customers that their check will be converted into an electronic transaction,

(2) the ability for a customer to opt-out of such conversion, and (3) the proper use of SEC Codes to indicate which method of check conversion was used to convert the paper check into an electronic item.

When sending remote deposit transactions through this API, please make sure to specify the proper SEC Code based on which conversion method was used in the merchant's business environment.

For more information regarding the the implementation of customer notification and opt-out rules, please visit the following website published by NACHA:

http://www.electronicpayments.org/businesses/bs.check-conversion.how.php

Note: If NACHA decides to audit a merchant or processor, part of the audit process may require providing proof of the authorization method (SEC Code) specified when the transaction was created. Failure to properly comply and provide proof of the authorization can result in fines for each transaction in violation, so it is important that you correctly indicate the correct SEC Code and maintain good records of your authorizations. Additional information about SEC Codes can be provided upon request.

Unique Transaction Identification

Proper communication between two separate transaction management applications (like this gateway and your software application) requires that both applications share a common, unique reference for each transaction in order for the two application to communicate intelligently.

For example, to modify or cancel a pending transaction, your software will need to supply a reference id that both systems recognize as uniquely identifying that transaction. Also, when a query command produces a result set containing multiple transactions, your software will need to know how to cross-reference each transaction in the query result with the associated transaction data in your software application.

For this reason, *Provider_TransactionID* is a required field when a Merchant submits transactions to SpeedChex for processing. Although a GUID or some similar universally unique ID is recommended for this value, the only requirement for this field is that the *Provider_TransactionID* value be unique for each transaction under each merchant.

Verification Using SpeedChex ExpressVerify

Merchants may choose to sign up for an optional bank account verification service called *SpeedChex Express Verify*. This service can report in real-time whether an account exists, or whether it is currently overdrawn, frozen or closed thus ascertaining whether a check is likely to be returned.

The service returns a 3-letter verification status which can be "POS" (postive) indicating the bank account is found and in good standing, "NEG" (negative) indicating the accound does not exist or is not in good standing, and "UNK" (unknown) indicating the bank account does not belong to a participating bank. The code "ERR" (error) can also result if technical problems occurred verifying the account.

Please see the document titled **SpeedChex Express Verify Response Codes** for a complete list of possible responses from the SpeedChex Express Verify system and their meanings.

Application Testing

Merchants can test the ***SpeedChex Web Service API*** by simply including an optional field called *TestMode* and setting the value of that field to "*On*". Test commands sent with "*TestMode=On*" will receive valid responses from the SpeedChex Gateway but the command will not actually be processed by the SpeedChex system.

The following information may be helpful when testing your application:

Test Merchant Gateway Credentials
MerchantID: 2001
Merchant_GateID: test
Merchant_GateKey: test

Test Bank Account that Passes SpeedChex Express Verify
Routing Number: 123123123
Account Number: <any number>

Test Bank Account that Fails SpeedChex Express Verify
Routing Number: 123123123
Account Number: 987654321

Please Note: Merchant ID 2001 is a demo account. If you send transactions using this Merchant ID and you do NOT set TestMode to "On", any information you transmit may be viewed by users running the SpeedChex demo. This includes names, addresses, phone numbers, and email addressees.

Data Security and Protection

Protecting the financial transaction data processed through SpeedChex is of utmost priority. This means not only implementing the highest levels of security standards in data encryption and system security, but also setting strict controls that limit authorized access to sensitive information.

Every Merchant is assigned a unique Merchant ID, GateID, and GateKey that must be kept confidential and will be required as part of each data packet sent to the SpeedChex Gateway. In addition, an IP filtering scheme may be implemented to ensure that command packets are only accepted from IP addresses registered by the Merchant.

Overview of the SpeedChex Gateway Command Process

Integrating the ***SpeedChex Web Service API*** into your software application is not difficult. The following is an overview of the major components of this task:

- **Data Gathering** - Merchants are responsible for collecting and submitting all data associated with a remote deposit command.

- **Submitting a Gateway Command –** Your software can use established SOAP 1.2 web service protocols to properly instantiate, populate and transmit a Transact_Command object for processing. The rules for constructing and sending the commands are defined in the next section of this document titled *General Implementation Rules and Specifications*.

- **Response Processing** - The SpeedChex Gateway will return a Transact_Response object after it receives and processes the command. The exact structure, format and meaning response object values will be based on the command issued as defined in the next section of this document titled *General Implementation Rules and Specifications*.

General Implementation Rules and Specifications

The ***SpeedChex Web Service API*** supports SOAP Version1.2 protocol for sending and receiving data through web service methods. MTOM is also support for sending and receiving binary files as necessary.

1. **Web Service Endpoint** – The following table shows the endpoint for the this web service and important proxy setup details:

Endpoint Address
https://www.speedchex.com/webservices/transact.svc
Comments
Please direct all proxy web service method calls and web reference or service reference proxy requests to this SSL secured Internet address. When using Microsoft Visual Studio 2008, create a Service Reference proxy to this URL to implement the advanced WS* features like MTOM. When using Microsoft Visual Studio 2005, create a Web Reference proxy to this URL, modify your project properties to support Web Services Enhancements (WSE). WSDL discovery can be accomplished by appending '?wsdl' to the end of this address.

2. **The Transact_Command Class** – The *Transact_Command* class provides the object definition for all properties that can be defined when sending a command to the SpeedChex Gateway.

   The following table lists the basic command template properties of the *Transact_Command* class which apply to all gateway commands:

The Transact_Command Class – Basic Command Template Properites

Field Name	Usage	Field Value Format Constraints
Command	Required	Set to the command you want the Transact Gateway to execute
CommandVersion	Required	Set to ***1.0*** for this API documentation revision.
TestMode	Optional	Set this value to ***On*** to test a command response.
Merchant_Credentials (assign a new *Transact_MerchantCredentials* object to this property with the following fields)		
.MerchantID	Required	Provided to Merchant
.GateID	Required	Provided to Merchant
.GateKey	Required	Provided to Merchant
<additional fields as required>		Based on the Command, you may be required to define additional fields to send in the Transact_Command object. These fields will be defined in the various sections of this document below dedicated to each specific command.

3. **Web Service Methods** – The following web service method definition(s) exist for this API:

Method Name	
ExecuteCommand (*Transact_Command* object) returns a *Transact_Response* object	
Usage	**Rules and Information**
Required	Create a new *Transact_Command* object, populate the fields in this object according to the rules defined in this API for the specific command that you wish to submit, and then call this method with your new *Transact_Command* object as its parameter. You will receive a *Transact_Response* object indicating whether the command succeeded or failed and any additional response information specifically related to the command issued.

4. **The Transact_Response Class** – In response to an *ExecuteCommand* web service method, the SpeedChex Gateway will always send a *Transact_Response* object indicating whether the command succeeded or failed and any additional information specifically related to the command issued.

   The following table defines the structure of the *Transact_Response* object with an explanation about the field values that will be returned in every response:

The Transact_Response Class

FieldName	Value Format Constraints	Max Length	Purpose
ResponseCode	A 3 digit code representing the reason for the command response. Please refer to the table in *Appendix A - Response Code Definitions*.	3	Provides a simple response code indicating success or reason for command failure.
Description	Please refer to the table in *Appendix A - Response Code Definitions*.	255	A brief explanation of the Response*Code* value
ErrorInformation	Additional information helpful to determine the source of an error. Please refer to the table in *Appendix A - Response Code Definitions*.	50	If the command failed, extra information about the error may be provided in this field.
ResponseData	Please see the documentation for the specific command to be issued for an explanation of the possible value(s) for this field.		This is a generic object that can take the form of any scalar or complex object called for by the command that is issued.
Transact_ReferenceID	A unique reference code assigned to each command.	30	This value can be used for as a unique transaction identifier or for support on any command.

Transaction Commands

Command: RemoteDepositBatch.CreateNewBatch

Description: Creates a new Remote Deposit Batch in a batch state of 'Hold for Review' for placing, managing and utlimately processing scanned checks for remote deposit. The following table defines the data field rules for this command:

Field Name	Usage	Field Value Format Constraints	Max Length
Command	Required	Set to ***RemoteDepositBatch.CreateBatch***	50
CommandVersion	Required	Set to ***1.0*** for this API documentation revision.	-
TestMode	Optional	Set this value to ***On*** to test a command response.	3
Merchant_Credentials	Required	Please refer to the Basic Command Template for details	-
BatchID	Required	A unique ID for this new Remote Deposit Batch.	50

Transaction Commands

Command: RemoteDepositBatch.AddTransaction

Description: Adds a new check to an existing Remote Deposit Batch. The following table defines the data field rules for this command:

Field Name	Usage	Field Value Format Constraints	Max Length
Command	Required	Set to ***RemoteDepositBatch.AddTransaction***	50
CommandVersion	Required	Set to ***1.0*** for this API documentation revision.	-
TestMode	Optional	Set this value to ***On*** to test a command response.	3
Merchant_Credentials	Required	Please refer to the Basic Command Template for details	-
Provider_TransactionID	Required	Unique ID assigned to this transaction by the Merchant. Required if this command is part of a batch *CommandArray*.	50
BatchID	Required	The ID of the new or existing Remote Deposit Batch to which this transaction is to be added.	50
CheckType	Required	Value must be **Personal**, **Business**, **Money Order**, **Cashiers Check**, or **Travellers Check**.	16
CheckNumber	Conditional *	The check number printed on the check.	25
RoutingNumber	Conditional *	9-digit ABA routing number on customer's check.	9
AccountNumber	Conditional *	Customer's bank account number.	30
Amount	Conditional *	The amount of the check. Do not include $ sign or comma.	-
Raw_MICR_Line	Required	The raw MICR line exactly as it was captured by the scanner. Please refer to the section of this document titled *Appendix B – Raw MICR Line Format* for special text formatting instructions.	100
CheckImage_Front	Required	A btye array of the binary image of the front of the check.	-
CheckImage_Back	Required	A btye array of the binary image of the back of the check.	-
Billing_CustomerID	Optional	The unique internal ID assigned to this customer.	50
Billing_CustomerName	Optional	Customer's Name	80
Billing_Company	Optional	Company Name	80
Billing_Address1	Optional	Customer's address	70
Billing_Address2	Optional	Customer's address	40
Billing_City	Optional	Customer's city	70
Billing_State	Optional	Accepts any valid state name or 2 letter abbreviation.	20
Billing_Zip	Optional	Customer's zip. (Format: ##### or #####-####)	10
Billing_Phone	Optional	Customer's phone. Any format, but must contain 10 digits	20
Billing_Email	Optional	Customer's email address. This field is required if the *SendEmailToCustomer* field value is **Yes**	80
Merchant_ReferenceID	Optional	The unique internal ID or invoice number assigned to this transaction by the merchant	40
Description	Optional	A description of this transaction	100
Run_ExpressVerify	Required	Value must be either **Yes** or **No**. .	3
Run_SmartScan	Required	Value must be either **Yes** or **No**.	3
SECCode	Conditional *	Values must be one of the following: **POP**, **BOC** or **ARC**. Please refer to the section of this document entitled *Important Rules Regarding Customer Notification and SEC Codes.*	3

* These fields are Required if the transaction is assigned to a batch whose state is set to 'Mark for Deposit'

Transaction Commands

Command: RemoteDepositBatch.UploadBatch

Description: Executes an array of *RemoteDepositBatch.AddTransaction* commands allowing you to add multiple transactions in "batch mode" to a new or existing Remote Deposit Batch. This command is equivalent to sending a *RemoteDepositBatch.CreateNewBatch* command and then sending multiple *RemoteDepositBatch.AddTransaction* commands each separately.

The following table defines the data field rules for this command:

Field Name	Usage	Field Value Format Constraints	Max Length
Command	Required	Set to ***RemoteDepositBatch.UploadBatch***	50
CommandVersion	Required	Set to ***1.0*** for this API documentation revision.	-
TestMode	Optional	Set this value to ***On*** to test a command response.	3
Merchant_Credentials	Required	Please refer to the Basic Command Template for details	-
BatchID	Required	The ID of a new or existing Remote Deposit Batch to which the transactions in the *CommandArray* will be assigned.	50
BatchState	Conditional	Value must be either **Hold for Review** or **Mark for Deposit**. Only required when creating a new Remote Deposit Batch. This field is ignored if the Remote Deposit Batch already exists.	25
LocationName	Optional	Any location category pre-defined in SpeedChex administration	50
DateScheduled	Conditional	The date to process this batch for deposit. Only required when creating a new batch and the *BatchState* is **Mark for Deposit**. This field is ignored otherwise. Format: "MM/DD/YYYY" (string)	10
CommandArray	Required	An array of new *RemoteDepositBatch.AddTransaction* commands. Please refer the *RemoteDepositBatch.AddTransaction* command for a list of required and optional fields and rules for data that must be submitted with each command in this array. **Special Note:** The following fields do not need to be defined for each *RemoteDepositBatch.AddTransaction* command object in this array because their values are already defined in the parent *RemoteDepositBatch.UploadBatch* command: • Provider_Credentials • Merchant_Credentials • BatchID • TestMode An error in any *RemoteDepositBatch.AddTransaction* command in this array will result in a general error for this parent command and a rejection of all transactions in the array.	-

Response: If the response to this *UploadBatch* command is an error caused by one of the *AddTransaction* commands in the *CommandArray*, the *Error_Information* field in r*eponse* object will prefix the error information value with "**CommandArray:**" For example, if the *CheckType* field was missing from a command in the *CommandArray*, the *ResponseCode* value would be "002" and the value of the *Error_Information* field would be "**CommandArray:CheckType**".

In addition, the *ResponseData* field of the reponse will contain the exact *Transact_Command* object from the *CommandArray* causing the error. The *Provider_TransactionID* field value in that command can then be used to help you find the transaction causing of the problem.

Transaction Commands

Command: RemoteDepositBatch.VoidBatch

Description: Cancels a Remote Deposit Batch that has not yet been sent to the Federal Reserve. All transaction data and images associated with the batch will be deleted permanently.. The following table defines the data field rules for this command:

Field Name	Usage	Field Value Format Constraints	Max Length
Command	Required	Set to ***RemoteDepositBatch.VoidBatch***	50
CommandVersion	Required	Set to ***1.0*** for this API documentation revision.	-
TestMode	Optional	Set this value to ***On*** to test a command response.	3
Merchant_Credentials	Required	Please refer to the Basic Command Template for details	-
BatchID	Required	A unique ID for this new Remote Deposit Batch.	50

Sample Client Code - Uploading a batch with the *RemoteDepositBatch.BatchUpload* command

Visual Basic.NET Sample Code

```
'Define the web service client object
Dim Transact_WebService As New Transact_WebServiceClient

'Create the new Transact_Command object to send to the web service
Dim BatchUpload_Command As New Transact_Command

'Create a new Transact_MerchantCredential objects and populate the gateway credential data
Dim MerchantCredentials As New Transact_MerchantCredentials
MerchantCredentials.MerchantID = "2001"
MerchantCredentials.GateID = "test"
MerchantCredentials.GateKey = "test"

'Specify the command to issue to the SpeedChex Gateway
BatchUpload_Command.Command = "RemoteDepositBatch.UploadBatch"
BatchUpload_Command.CommandVersion = "1.0"

'Assign the gateway credential objects to the command
BatchUpload_Command.Merchant_Credentials = MerchantCredentials
BatchUpload_Command.Provider_Credentials = ProviderCredentials

'Define the information specific to the Remote Deposit Batch being created/uploaded
'    Suggestion: create a date based BatchID or use a GUID to ensure a unique ID for each new batch.
BatchUpload_Command.BatchID = "2001_" & Today.ToString("yyyy_MM_dd") & "_001"
BatchUpload_Command.BatchState = "Mark for Deposit"
BatchUpload_Command.DateScheduled = Today.Date

'Create a new generic array that will hold the batch of RemoteDepositBatch.AddTransaction command objects
Dim myCommandArray As New ArrayList

'Iterate through your database to populate and add each "AddTransaction" command object to the array
For Each myDataRow In MyDataTable
        'Create a new Transact_Command object to put into the CommandArray
        Dim AddCheck_Command As New Transact_Command

        'Specify the RemoteDepositBatch.AddTransaction as the command to execute when this CommandArray is processed
        AddCheck_Command.Command = "RemoteDepositBatch.AddTransaction"
        AddCheck_Command.CommandVersion = "1.0"

        'Define the unique ID you have assigned to this transaction internally for later status tracking, etc.
        AddCheck_Command.Provider_TransactionID = myDataRow.myUniqueTransactionID

        'Assign known or captured check information to command
        AddCheck_Command.RoutingNumber = myDataRow.RoutingNumber
        AddCheck_Command.AccountNumber = myDataRow.AccountNumber
        AddCheck_Command.Amount = myDataRow.Amount
        '  ...  etc. for all optional or required check information fields

        'Assign the byte array of captured check images
        AddCheck_Command.CheckImage_Front = System.IO.File.ReadAllBytes("c:\FrontOfCheck.tif")
        AddCheck_Command.CheckImage_Back = System.IO.File.ReadAllBytes("c:\BackOfCheck.tif")

        'Add this command to the CommandArray object
        myCommandArray.Add(AddCheck_Command)
Next

'When the batch (CommandArray) of commands is complete, assigned array to the BatchUpload_Command object
BatchUpload_Command.CommandArray = myCommandArray.ToArray


***** Viisual Basic.Net Sample Code Continued on Next Page *****
```

Sample Client Code - Uploading a batch with the *RemoteDepositBatch.BatchUpload* command

Visual Basic.NET Sample Code (continued…)

```
'*************************************************************************************************
'*   Ready to submit the BatchUpload_Command to the SpeedChex Gateway
'*************************************************************************************************

'Create a Transact_Response object to receive the response from the gateway
Dim BatchUpload_Response As Transact_Response

'Execute the Transact Gateway command
BatchUpload_Response = Transact_WebService.ExecuteCommand(BatchUpload_Command)


'Parse the response
If BatchUpload_Response.ResponseCode = "000" Then
    'Write code related to successful response here

Else
    'Batch upload failed.  Parse the reason
    Select Case BatchUpload_Response.ResponseCode
        Case "001"
            'Write error handling code here
        Case "002"
            'Write error handling code here
            'Add additional parsing cases to handle all possible responses
            '...
            '   additional Case statement as necessary
            '...
    End Select
End If
```

Sample Client Code - Uploading a batch with the *RemoteDepositBatch.BatchUpload* command

C#.NET Sample Code

```
//Define the web service client object
Transact_WebServiceClient Transact_WebService = new Transact_WebServiceClient();

//Create the new Transact_Command object to send to the web service
Transact_Command BatchUpload_Command = new Transact_Command();

//Create a new Transact_MerchantCredential objects and populate the gateway credential data
Transact_MerchantCredentials MerchantCredentials = new Transact_MerchantCredentials();
MerchantCredentials.MerchantID = "2001";
MerchantCredentials.GateID = "test";
MerchantCredentials.GateKey = "test";

//Specify the command to issue to the SpeedChex Gateway
BatchUpload_Command.Command = "RemoteDepositBatch.UploadBatch";
BatchUpload_Command.CommandVersion = "1.0";

//Assign the gateway credential objects to the command
BatchUpload_Command.Merchant_Credentials = MerchantCredentials;
BatchUpload_Command.Provider_Credentials = ProviderCredentials;

//Define the information specific to the Remote Deposit Batch being created/uploaded
// Suggestion: create a date based BatchID or use a GUID to ensure a unique ID for each new batch.
BatchUpload_Command.BatchID = "2001_" + Today.ToString("yyyy_MM_dd") + "_001";
BatchUpload_Command.BatchState = "Mark for Deposit";
BatchUpload_Command.DateScheduled = Today.Date;

//Create a new generic array that will hold the batch of RemoteDepositBatch.AddTransaction command objects
ArrayList myCommandArray = new ArrayList();

//Iterate through your database to populate and add each "AddTransaction" command object to the array
foreach ( myDataRow in MyDataTable) {
   //Create a new Transact_Command object to put into the CommandArray
   Transact_Command AddCheck_Command = new Transact_Command();

   //Specify the RemoteDepositBatch.AddTransaction as the command to execute when this CommandArray is processed
   AddCheck_Command.Command = "RemoteDepositBatch.AddTransaction";
   AddCheck_Command.CommandVersion = "1.0";

   //Define the unique ID you have assigned to this transaction internally for later status tracking, etc.
   AddCheck_Command.Provider_TransactionID = myDataRow.myUniqueTransactionID;

   //Assign known or captured check information to command
   AddCheck_Command.RoutingNumber = myDataRow.RoutingNumber;
   AddCheck_Command.AccountNumber = myDataRow.AccountNumber;
   AddCheck_Command.Amount = myDataRow.Amount;
   // ...  etc. for all optional or required check information fields
   AddCheck_Command.SECCode = "BOC";

   //Assigned the byte array of captured check images
   AddCheck_Command.CheckImage_Front = System.IO.File.ReadAllBytes("c:\\FrontOfCheck.tif");
   AddCheck_Command.CheckImage_Back = System.IO.File.ReadAllBytes("c:\\BackOfCheck.tif");

   //Add this command to the CommandArray object
   myCommandArray.Add(AddCheck_Command);
}

//When the batch (CommandArray) of commands is complete, assigned array to the BatchUpload_Command object
BatchUpload_Command.CommandArray = myCommandArray.ToArray;


***** C#.Net Sample Code Continued on Next Page *****
```

Sample Client Code - Uploading a batch with the *RemoteDepositBatch.BatchUpload* command

C#.NET Sample Code (continued…)

```
//*****************************************************************************************
//* Submit the BatchUpload_Command to the SpeedChex Transact Gateway
//*****************************************************************************************

//Create a Transact_Response object to receive the response from the gateway
Transact_Response BatchUpload_Response;

//Execute the Transact Gateway command
BatchUpload_Response = Transact_WebService.ExecuteCommand(BatchUpload_Command);

//Parse the response
if (BatchUpload_Response.ResponseCode == "000") {
            //Write code related to successful response here
}
else {
   //Batch upload failed. Parse the reason
   switch (BatchUpload_Response.ResponseCode) {
      case "001":
            //Write error handling code here
           break;
       case "002":
            //Write error handling code here
           break;

       //Add additional parsing cases to handle all possible responses
  }
}
```

Appendix A – Response Code Definitions

Response Code	Description	Contents of the *ErrorInformation* Field
GATEWAY COMMAND SUCCESS		
000	Command Successful. Approved.	
GATEWAY COMMAND ERRORS		
100	Invalid Gateway Credentials	
101	Invalid Gateway Command	
102	Duplicate Command Not Processed	Transact_ReferenceID of the original Command
103	Transaction Cannot Be Modified	Transaction Status
104	Batch Cannot Be Modified	Batch Status
105	Invalid Transact_ReferenceID	
106	Invalid BatchID	
107	Non-Unique Reference/Transaction ID	
108	Invalid Reference/Transaction ID	
109	Invalid Source IP	
110	Invalid Value In Message	
INPUT DATA VALIDATION ERRORS		
150	Required Field Missing	Field Name
151	Field Value Is Not Valid	Field Name
152	Field Value Exceeds Maximum Length	Field Name
PAYMENT ACCOUNT VERIFICATION FAILURES		
200	Failed AVS	
201	Failed CVN	
202	Failed Express Verify	
203	Invalid Credit Card Number	
204	No Such Card Issuer	
205	Expired Card	
206	Invalid Expiration Date	
208	Call Issuer for Further Information	
209	Invalid Routing Number	
210	Invalid Bank Account Number	
211	Invalid PIN	
212	Invalid PaymentKey	
PAYMENT ACCOUNT DECLINES		
300	Transaction was Declined by Processor	
301	Transaction was Rejected by Gateway	
302	No Card Number on File with Issuer	
304	Invalid Account Type	
305	Account Closed	
306	Account Inactive	

Response Code Defintions Continued on Next Page…

Appendix A – Response Code Definitions

Response Code	Description	Contents of the ErrorInformation Field
PAYMENT ACCOUNT DECLINES (continued…)		
307	Account Frozen	
309	Insufficient Funds	
310	Over Limit	
311	Do Not Honor	
312	Transaction Not Allowed	
313	Invalid for Debit	
314	Invalid for Credit	
315	Customer Opt Out	
316	Customer Advises Not Authorized	
317	Manual Key Not Allowed	
318	Duplicate Transaction at Processor	
FRAUD DECLINES		
400	Pick Up Card	
401	Lost Card	
402	Stolen Card	
403	Fraudulant Card	
404	Excessive Declines From Same Source	
405	Excessive PIN Attempts	
406	Excessive Purchase Frequency	
MERCHANT DIRECTIVES FROM PROCESSOR		
500	Declined - Stop All Recurring Payments	
501	Declined - Update Cardholder Data Available	
502	Declined - Further Instructions Available	Instructions
503	Declined - Call Processor for Voice Authorization	
504	Declined - Call Processor for Fraud Instructions	
PROCESSOR ADMINISTRATIVE ERRORS		
600	Internal Gateway Error	
601	Internal Processor Error	
602	Communication Error with Issuer	
603	Communication Error with Processor	
604	Processor Feature Not Available	
605	Processor Format Error	
606	Invalid Terminal Number	
607	Merchant Not Setup	
608	Merchant Account is Inactive	
609	Invalid Merchant Configuration	
610	Invalid Payment Method for Merchant	
611	Unsupported Card Type	
OTHER		
999	Contact Support Representative	

Appendix B – Raw MICR Line Format

The raw MICR line at the bottom of a check contains special characters that cannot be transmitted in a string field. As a result, the value supplied for the Raw MICR Line must use the following letters to represent the following potential special characters found on the bottom of the check:

T = Transit Symbol
U = OnUs Symbol
B = Blank (space)
D = Dash Symbol
A = Amount Symbol
E = Error or Unknown character

Implementation Support

If you need help understanding this documentation or with any of the details of integrating the ***SpeedChex Web Service API*** into your application, please do not hesitate to contact our support staff by email at support@speedchex.com.

If you need to speak to a support team member, please put your name and phone number on the email and the best time to call.